

МИХАИЛ АБРАМЯН

Visual C#

НА ПРИМЕРАХ

РАЗРАБОТКА ПРИЛОЖЕНИЙ
В MICROSOFT VISUAL C# 2005/2008

ИСПОЛЬЗОВАНИЕ БИБЛИОТЕКИ КЛАССОВ
WINDOWS FORMS

РЕАЛИЗАЦИЯ УДОБНОГО
И НАДЕЖНОГО ИНТЕРФЕЙСА

ОБСУЖДЕНИЕ ТИПИЧНЫХ ОШИБОК
И СПОСОБОВ ИХ ИСПРАВЛЕНИЯ

+CD

bhy®

Михаил Абрамян

Visual C#

НА ПРИМЕРАХ

Санкт-Петербург

«БХВ-Петербург»

2008

УДК 681.3.068+800.92Visual C#
ББК 32.973.26-018.1
А16

Абрамян М. Э.

А16 Visual C# на примерах. — СПб.: БХВ-Петербург, 2008. — 496 с.: ил.
+ CD-ROM

ISBN 978-5-9775-0266-5

Книга содержит подробное описание 32 проектов, демонстрирующих различные аспекты создания Windows-приложений для платформы .NET Framework в среде Microsoft Visual C# 2005/2008. Рассматриваются оптимальные приемы разработки программ, управляемых событиями, механизм обработки исключений, особенности консольных и MDI-приложений. Детально описываются основные компоненты библиотеки Windows Forms и классы, входящие в графическую библиотеку GDI+. Демонстрируются приемы работы с клавиатурой и мышью, а также дополнительные возможности .NET-приложений, в том числе реализация режима перетаскивания drag & drop, работа с реестром Windows и др. На компакт-диске содержатся исходные тексты проектов, описанных в книге.

Для программистов

УДК 681.3.068+800.92Visual C#
ББК 32.973.26-018.1

Группа подготовки издания:

Главный редактор	<i>Екатерина Кондукова</i>
Зам. главного редактора	<i>Игорь Шишигин</i>
Зав. редакцией	<i>Григорий Добин</i>
Редактор	<i>Анна Кузьмина</i>
Компьютерная верстка	<i>Натальи Караваевой</i>
Корректор	<i>Виктория Пиотровская</i>
Дизайн серии	<i>Игоря Цырульникова</i>
Оформление обложки	<i>Елены Беляевой</i>
Зав. производством	<i>Николай Тверских</i>

Лицензия ИД № 02429 от 24.07.00. Подписано в печать 25.06.08.

Формат 70×100^{1/16}. Печать офсетная. Усл. печ. л. 39,99.

Тираж 2000 экз. Заказ №

"БХВ-Петербург", 194354, Санкт-Петербург, ул. Есенина, 5Б.

Отпечатано с готовых диапозитивов
в ГУП "Типография "Наука"
199034, Санкт-Петербург, 9 линия, 12

ISBN 978-5-9775-0266-5

© Абрамян М. Э., 2008
© Оформление, издательство "БХВ-Петербург", 2008

Оглавление

Предисловие	3
ЧАСТЬ I	7
Глава 1. Разработка программ в среде Microsoft Visual Studio .NET	9
1.1. Создание, сохранение и открытие проекта.....	9
1.2. Добавление к проекту новой формы и размещение в форме нового компонента	12
1.3. Настройка свойств форм и компонентов	16
1.4. Определение обработчиков событий	19
1.5. Внесение изменений в текст программы.....	20
1.6. Запуск приложения	23
Глава 2. Консольное приложение: проект DISKINFO	25
2.1. Создание консольного приложения.....	25
Комментарии	27
2.2. Получение информации о текущем диске	28
Комментарии	29
2.3. Использование параметров командной строки	32
Комментарий	34
Глава 3. Обработка исключений: проект EXCER.....	35
3.1. Обработка конкретного исключения и групп исключений	35
Комментарии	39
3.2. Обработка любого исключения	40
Комментарий	41
3.3. Повторное возбуждение обработанного исключения.....	42
Комментарии	42
Глава 4. События: проект EVENTS.....	44
4.1. Связывание события с обработчиком	44
Комментарии	46

4.2. Отключение обработчика от события	48
Комментарии	49
4.3. Подключение к событию другого обработчика	51
Комментарии	53
Глава 5. Формы: проект FORMS	55
5.1. Настройка визуальных свойств форм. Открытие форм в обычном и модальном режиме.....	55
Комментарии	58
5.2. Контроль за состоянием подчиненной формы. Воздействие подчиненной формы на главную	60
Комментарии	60
5.3. Компоненты, подстраивающиеся под размер окна.....	61
Комментарии	62
5.4. Модальные и обычные кнопки диалогового окна	63
Комментарии	64
5.5. Установка активного компонента формы	65
5.6. Запрос на подтверждение закрытия формы	66
Комментарии	68
Глава 6. Совместное использование обработчиков событий и работа с клавиатурой: проект CALC.....	70
6.1. Обработчик событий для нескольких компонентов	70
6.2. Вычисления с контролем правильности исходных данных.....	72
Комментарии	73
6.3. Простейшие приемы ускорения работы с помощью клавиатуры	74
6.4. Использование обработчика событий от клавиатуры	75
Комментарии	77
6.5. Контроль за изменением исходных данных	77
Глава 7. Курсоры и значки: проект CURSORS.....	79
7.1. Использование стандартных курсоров	79
Комментарии	81
7.2. Установка курсора для формы и индикация режима ожидания с помощью курсора.....	82
Комментарии	83
7.3. Подключение к проекту новых курсоров и их сохранение в виде внедренных ресурсов	84
Комментарии	85

7.4. Работа со значками.....	86
Комментарии	87
7.5. Размещение значка в области уведомлений	87
Комментарии	88
Глава 8. Поля ввода: проект TEXTBOXES	90
8.1. Дополнительное выделение активного поля ввода	90
Комментарии	92
8.2. Управление порядком обхода полей в форме	93
Комментарий	95
8.3. Блокировка выхода из незаполненного поля ввода	96
Комментарии	96
8.4. Информирование пользователя о возникшей ошибке.....	97
Комментарий	97
8.5. Предоставление дополнительной информации об ошибке	98
Комментарий	99
8.6. Проверка ошибок на уровне формы.....	99
Комментарий	100
Глава 9. Цвета: проект COLORS	101
9.1. Определение цвета как комбинации четырех цветовых составляющих. Ползунки.....	101
Комментарии	104
9.2. Инвертирование цветов и вывод цветовых констант	106
Комментарии	107
9.3. Отображение оттенков серого цвета	108
9.4. Вывод цветовых имен.....	108
Комментарии	109
9.5. Связывание компонентов с подписями к ним	111
Комментарий	111
9.6. Привязка компонентов.....	111
Комментарии	112
Глава 10. Обработка событий от мыши: проект MOUSE	114
10.1. Перетаскивание с помощью мыши. Настройка z-порядка компонентов.....	114
Комментарии	116
10.2. Изменение размеров с помощью мыши.....	119
Комментарии	121

10.3. Использование дополнительных курсоров.....	122
10.4. Обработка ситуации с одновременным нажатием двух кнопок мышь: первый вариант	123
Комментарий	125
10.5. Обработка ситуации с одновременным нажатием двух кнопок мышь: второй вариант	125
Комментарий	126
10.6. Обработка ситуации с одновременным нажатием двух кнопок мышь: третий вариант	126
Комментарии	128
10.7. Перетаскивание компонентов любого типа. Поиск и замена в текстах программ	129
Комментарии	130
Глава 11. Перетаскивание (drag & drop): проект ZOO	133
11.1. Перетаскивание меток по форме	133
Комментарии	135
11.2. Перетаскивание меток в поля ввода	137
11.3. Взаимодействие меток при их перетаскивании друг на друга	139
Комментарии	140
11.4. Действия в случае перетаскивания на недопустимый приемник.....	141
Комментарий	141
11.5. Дополнительное выделение источника и приемника в ходе перетаскивания	142
Комментарии	143
11.6. Настройка вида курсора в режиме перетаскивания.....	144
Комментарии	144
11.7. Информация о текущем состоянии программы. Кнопки с изображениями	145
Комментарии	147
11.8. Восстановление исходного состояния	148
Комментарий	149
Глава 12. Работа с датами и временем: проект CLOCK.....	151
12.1. Отображение текущего времени.....	151
Комментарий	152
12.2. Реализация возможностей секундомера	153
Комментарий	157

12.3. Альтернативные варианты выполнения команд с помощью мыши.....	158
Комментарий	159
12.4. Отображение текущего состояния часов и секундомера на панели задач.....	159
Глава 13. Просмотр изображений: проект IMGVIEW	161
13.1. Добавление в окно <i>Toolbox</i> новых компонентов	161
Комментарий	162
13.2. Просмотр изображений из файлов на текущем диске.....	162
Комментарии	166
13.3. Стыковка компонентов и ее особенности.....	170
Комментарий	171
13.4. Возможность смены диска	172
Комментарии	174
13.5. Настройка режима просмотра изображений	174
Комментарии	176
13.6. Сохранение в реестре Windows информации о состоянии программы ...	176
Комментарии	177
13.7. Восстановление из реестра Windows информации о состоянии программы	179
Комментарии	181
ЧАСТЬ II.....	183
Глава 14. Работа с графическими файлами, рисование тонким пером: проект PNGEDIT1	185
14.1. Создание, сохранение и загрузка графических файлов	185
Комментарии	192
14.2. Отслеживание текущих координат изображения.....	196
14.3. Рисование тонким пером.....	197
Комментарии	199
14.4. Очистка изображения	200
Комментарий	201
Глава 15. Цветное перо и прямые линии: проект PNGEDIT2	202
15.1. Рисование цветным пером	202
Комментарии	204
15.2. Второй режим рисования: прямые линии.....	205
Комментарии	209

Глава 16. Прямоугольники и эллипсы, режим прозрачности: проект PNGEDIT3.....	211
16.1. Настройка фонового цвета	211
Комментарии	213
16.2. Третий режим рисования: прямоугольники	213
Комментарии	216
16.3. Рисование эллипсов	217
Комментарии	219
16.4. Рисование прозрачных фигур	220
Глава 17. Дополнительные графические возможности: проект PNGEDIT4.....	223
17.1. Рисование квадратов и окружностей	223
Комментарии	225
17.2. Отмена предыдущей операции	225
Комментарий	227
17.3. Задание цветов с помощью пипетки	228
Комментарии	229
17.4. Четвертый режим рисования: добавление в рисунок текста	229
Комментарии	232
17.5. Настройка стиля изображения линии	233
Комментарии	235
Глава 18. Меню и работа с текстовыми файлами: проект TXTEDIT1.....	236
18.1. Создание меню	236
Комментарии	239
18.2. Сохранение текста в файле	240
Комментарии	241
18.3. Очистка области редактирования и открытие нового файла.....	242
Комментарии	244
18.4. Запрос о сохранении изменений, внесенных в текст	245
Комментарии	246
Глава 19. Дополнительные возможности меню, настройка цвета и шрифта: проект TXTEDIT2.....	248
19.1. Установка начертания символов (команды меню — флажки).....	248
Комментарии	250

19.2. Установка выравнивания текста (команды меню — переключатели).....	251
Комментарии	252
19.3. Установка цвета символов и фона (команды меню третьего уровня и окно диалога <i>Цвет</i>)	253
19.4. Установка свойств шрифта с помощью окна диалога <i>Шрифт</i>	254
Комментарии	257
Глава 20. Команды редактирования, контекстное меню: проект TXTEDIT3	258
20.1. Команды редактирования	258
Комментарии	260
20.2. Выделение недоступных команд редактирования. Работа с буфером обмена.....	261
Комментарий	262
20.3. Создание контекстного меню.....	263
Глава 21. Панель инструментов: проект TXTEDIT4	265
21.1. Создание панели инструментов с кнопками быстрого доступа. Добавление изображений к пунктам меню	265
Комментарий	268
21.2. Размещение на панели инструментов кнопок-флажков и кнопок-переключателей	269
Комментарии	272
Глава 22. Статусная панель и подсказки: проект TXTEDIT5	273
22.1. Использование статусной панели.....	273
Комментарии	275
22.2. Недоступные кнопки быстрого доступа.....	275
22.3. Скрытие панелей.....	275
22.4. Вывод подсказок на статусную панель	276
Комментарии	278
Глава 23. Форматирование документа: проект TXTEDIT6	280
23.1. Замена компонента <i>TextBox</i> на компонент <i>RichTextBox</i>	280
Комментарии	283
23.2. Корректировка состояния кнопок быстрого доступа и команд меню при изменении текущего формата	284
Комментарий	286
23.3. Настройка свойств абзаца	286
Комментарии	289

23.4. Отображение текущей строки и столбца	289
Комментарий	290
23.5. Загрузка и сохранение текста без форматных настроек	291
Комментарии	292
ЧАСТЬ III	295
Глава 24. Флажки и группы флажков: проект CHKBOXES	297
24.1. Установка флажков и проверка их состояния	297
Комментарии	300
24.2. Глобальная установка флажков	301
24.3. Использование флажков, принимающих три состояния	302
Глава 25. Выпадающие и обычные списки: проект LISTBOX1	306
25.1. Создание и использование выпадающих списков	306
Комментарии	308
25.2. Список: добавление и удаление элементов	308
Комментарии	311
25.3. Дополнительные операции над списком	312
Комментарии	315
25.4. Выполнение операций над списком с помощью мыши	315
Комментарии	317
Глава 26. Списки с множественным выделением и графические списки: проект LISTBOX2	319
26.1. Списки с множественным выделением	319
Комментарии	322
26.2. Обработка выделенных элементов	323
Комментарии	326
26.3. Графические списки	327
Комментарии	329
Глава 27. MDI-приложение: проект JPEGVIEW	331
27.1. Открытие и закрытие дочерних форм в MDI-приложении	331
Комментарии	335
27.2. Стандартные действия над дочерними формами	336
Комментарии	338
27.3. Добавление в меню списка открытых дочерних форм	339
Комментарий	340
27.4. Одновременное закрытие всех дочерних форм	340
Комментарий	341

27.5. Масштабирование изображения.....	342
Комментарий	343
27.6. Автоматическая корректировка размера дочерних форм.....	343
Комментарии	345
27.7. Дополнительные средства управления дочерними формами	345
Комментарии	349
27.8. Прокрутка изображения с помощью клавиатуры	349
Комментарий	351
Глава 28. Таблица с текстовыми данными: проект FONTVIEW	352
28.1. Получение списка доступных шрифтов	352
Комментарии	353
28.2. Отображение символов шрифта в таблице.....	354
Комментарии	357
28.3. Настройка стиля для выбранного шрифта.....	358
Комментарии	360
28.4. Просмотр текущего символа в увеличенном виде.....	361
Комментарии	363
Глава 29. Таблица с графическими данными: проект ICONVIEW	365
29.1. Извлечение значков из файлов.....	365
Комментарии	368
29.2. Загрузка значков и их отображение в таблице	369
Комментарии	371
29.3. Очистка таблицы.....	373
29.4. Отображение дополнительной информации о выбранном значке	374
Комментарии	377
29.5. Переключение между увеличенным и обычным изображением иконки.....	377
29.6. Сохранение значка в виде ico- или png-файла	378
Глава 30. Приложение с заставкой: проект TRIGFUNC	381
30.1. Формирование таблицы значений тригонометрических функций.....	381
Комментарии	384
30.2. Отображение окна-заставки при загрузке программы	386
Комментарии	388
30.3. Использование окна-заставки в качестве информационного окна	389
30.4. Вывод в окне-заставке информации о ходе загрузки программы	390
Комментарии	392

30.5. Досрочное завершение программы.....	393
Комментарии	393
30.6. Возможность перемещения окна-заставки.....	394

Глава 31. Обработка наборов данных: проект STUD1.....396

31.1. Определение класса с описанием структуры данных и связывание этой структуры с таблицей в режиме дизайнера.....	396
Комментарий	402
31.2. Проверка правильности данных на уровне ячейки таблицы.....	403
Комментарии	405
31.3. Проверка правильности данных на уровне строки таблицы	406
Комментарий	407
31.4. Дополнительная настройка столбцов таблицы, использование выпадающих списков для текстовых полей	408
Комментарии	409
31.5. Использование XML-сериализации для сохранения и загрузки наборов данных	410
Комментарии	417
31.6. Дополнительные средства навигации и редактирования для таблицы с набором данных	418
Комментарий	420
31.7. Автоматизация действий при добавлении нового элемента в набор данных	420
Комментарии	422

Глава 32. Дополнительные возможности, связанные с обработкой наборов данных: проект STUD2.....423

32.1. Использование набора вкладок и добавление таблицы с подчиненными данными.....	423
Комментарии	430
32.2. Связывание с данными компонентов, не являющихся таблицами.....	431
Комментарии	432
32.3. Выпадающие списки, связанные с числовыми полями	434
Комментарии	436
32.4. Вычисляемые столбцы	437
Комментарии	439
32.5. Сортировка данных.....	439
Комментарии	442
32.6. Поиск по шаблону.....	442
Комментарии	445

Глава 33. Создание компонентов во время выполнения программы: проект HTOWERS	447
33.1. Создание начальной позиции.....	447
Комментарий.....	449
33.2. Перерисовка башни при изменении количества блоков.....	449
Комментарии.....	450
33.3. Перетаскивание блоков на новое место.....	451
Комментарий.....	454
33.4. Восстановление начальной позиции и подсчет числа перемещений блоков.....	454
Комментарий.....	456
33.5. Проверка решения задачи.....	456
33.6. Выполнение задачи в демо-режиме.....	457
Комментарий.....	459
33.7. Настройка идентификационных данных приложения.....	460
Комментарии.....	461
ПРИЛОЖЕНИЯ	463
Приложение 1. Краткий словарь используемых терминов	465
Приложение 2. Описание компакт-диска	473
Литература	476
Предметный указатель	477

*Памяти моего учителя профессора
Игоря Борисовича Симоненко (1935—2008)*

Предисловие

Книга, предлагаемая вашему вниманию, содержит подробные указания по разработке 32 примеров Windows-приложений для платформы .NET Framework. При создании приложений используется язык Visual C#. Предполагается, что в качестве интегрированной среды используется среда Microsoft Visual Studio .NET 2005 или 2008.

Каждый из примеров книги посвящен определенной теме, указанной в его названии, однако при описании процесса разработки проектов и в сопутствующих комментариях приводятся дополнительные сведения, что позволяет охватить очень широкий диапазон вопросов, связанных с созданием Windows-приложений. При этом особое внимание уделяется оптимальным приемам разработки приложений, управляемых событиями, и эффективному применению компонентов библиотеки Microsoft Windows Forms. Подробно обсуждаются средства, обеспечивающие удобный и надежный диалог программы с пользователем. Кроме того, рассматриваются типичные ошибки, возникающие при использовании различных классов библиотеки Windows Forms, и указываются способы их исправления.

Книга ориентирована на читателя, владеющего основами программирования. Желательно также знакомство с языком C# (на уровне знания системы базовых типов и управляющих операторов). Для ознакомления с языком C# можно воспользоваться, например, книгой [4], являющейся, к тому же, мастерски написанным учебником по программированию. Глубокого знания *объектной модели* платформы .NET не требуется. Основные термины, связанные с объектной моделью .NET, кратко объясняются в словаре, включенном в книгу в качестве приложения 1.

В версии .NET 2.0 библиотека Microsoft Windows Forms подверглась существенной переработке, поэтому многие возможности, описанные в книге, будут

недоступны для более ранних версий .NET (в частности, для версии .NET 1.1 и ориентированной на нее среды Microsoft Visual Studio .NET 2003). В версиях .NET 3.0 и 3.5 библиотека Microsoft Windows Forms практически не изменилась. Для того чтобы сохранить совместимость с широко распространенной версией .NET 2.0 и средой Microsoft Visual Studio .NET 2005, в книге не используются возможности, добавленные в язык Visual C#, начиная с версии 3.0. Следует отметить, что эти возможности (в частности, технология LINQ — Language-Integrated Query — технология запросов, интегрированных в язык программирования) в основном ориентированы на приложения, связанные с обработкой баз данных, которые в книге не рассматриваются.

Кратко опишем содержимое книги.

Глава 1 знакомит читателя с основными приемами работы в среде Microsoft Visual Studio .NET. Пример DISKINFO (см. главу 2) посвящен особенностям разработки консольных .NET-приложений. Пример EXCER (см. главу 3), также являющийся консольным приложением, демонстрирует различные аспекты обработки исключительных ситуаций.

Далее следует группа из 10 примеров, посвященных основам разработки графических .NET-приложений. В примере EVENTS (см. главу 4) рассматриваются вопросы, связанные с событиями и их обработкой. Пример FORMS (см. главу 5) посвящен формам и их взаимодействию в рамках одного приложения. Пример CALC (см. главу 6) знакомит с возможностью совместного использования обработчика для нескольких событий, а также описывает различные приемы работы с клавиатурой. Пример CURSORS (см. главу 7) показывает, как использовать в приложениях курсоры и значки (пиктограммы). Пример TEXT-BOXES (см. главу 8) знакомит с различными возможностями полей ввода, а также описывает средства, связанные с проверкой введенных данных и выводом сообщений об ошибках. Пример COLORS (см. главу 9) знакомит с типами, предназначенными для работы с цветом, и с компонентами, обеспечивающими прокрутку данных. Пример MOUSE (см. главу 10) посвящен событиям, связанным с манипулятором "мышь". В примере ZOO (см. главу 11) описывается, как реализовать режим перетаскивания drag & drop. Пример CLOCK (см. главу 12) знакомит с типами, связанными с датами и временем. Наконец, пример IMGVIEW (см. главу 13) посвящен компонентам, обеспечивающим навигацию по файловой системе; кроме того, в нем показано, как использовать реестр Windows для хранения информации о текущих настройках приложения.

Следующая группа из 10 примеров посвящена реализации двух полнофункциональных редакторов: графического и текстового. Разработка графического редактора (с применением средств графической библиотеки GDI+) проводится в серии примеров PNGEDIT. В примере PNGEDIT1 (см. главу 14) рассматрива-

ются основные действия с графическими файлами и приводится простейшая реализация рисования тонким одноцветным пером. Пример PNGEDIT2 (см. главу 15) показывает, как реализовать в редакторе рисование цветным пером произвольной толщины и наглядное рисование прямых линий. В примере PNGEDIT3 (см. главу 16) редактор дополняется возможностью рисования замкнутых фигур (прямоугольников и эллипсов) в обычном режиме и в режиме прозрачности, т. е. без заливки внутренности фигуры фоновым цветом. В примере PNGEDIT4 (см. главу 17) графический редактор снабжается некоторыми полезными дополнительными возможностями, среди которых следует отметить отмену последней графической операции, настройку цветов с помощью инструмента "пипетка", добавление к изображению текста и настройку стиля линии.

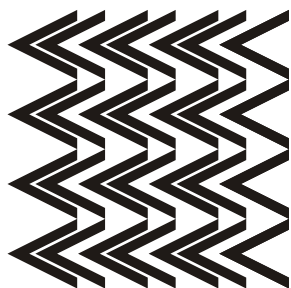
Разработка текстового редактора проводится в серии примеров TXTEDIT. В примере TXTEDIT1 (см. главу 18) описывается создание простейшего меню приложения и реализуются основные действия с текстовыми файлами. В примере TXTEDIT2 (см. главу 19) рассматриваются дополнительные возможности, связанные с меню (команды-флажки и команды-переключатели). В примере TXTEDIT3 (см. главу 20) реализуются стандартные команды редактирования и рассматривается работа с контекстным меню. Пример TXTEDIT4 (см. главу 21) описывает возможности панели инструментов, а пример TXTEDIT5 (см. главу 22) — статусной панели. Наконец, в примере TXTEDIT6 (см. главу 23) описываются действия по преобразованию созданного текстового редактора в форматированный, т. е. позволяющий по-разному форматировать различные фрагменты текста и сохранять форматные настройки вместе с текстом. При этом подробно рассматриваются возможности компонента RichTextBox.

Завершающая группа из 10 примеров посвящена углубленному изучению некоторых важных компонентов библиотеки Microsoft Windows Forms. В примере CHKBOXES (см. главу 24) описываются различные возможности компонентов-флажков, а также групп флажков; в серии из двух примеров LISTBOX1–LISTBOX2 (см. главы 25 и 26) рассматриваются выпадающие и обычные списки. Пример JPEGVIEW (см. главу 27) посвящен разработке приложения с многодокументным интерфейсом (MDI-приложения). В следующих пяти примерах основную роль играет компонент-таблица DataGridView. Пример FONTVIEW (см. главу 28) посвящен приемам работы со шрифтами и их семействами; в нем таблица используется для отображения символов различных шрифтов. В примере ICONVIEW (см. главу 29) реализуется возможность чтения значков из exe- и dll-файлов; в нем таблица DataGridView используется для отображения графической информации. В этом примере также показывается, как импортировать стандартные функции Win32 API в .NET-приложение, и приводится пример создания объектной оболочки для импортированной функции. Пример

TRIGFUNC (см. главу 30) показывает, как создать окно-заставку с особыми свойствами; кроме того, в нем содержатся дополнительные сведения о настройке свойств таблицы `DataGridView`. Следующие два примера `STUD1-STUD2` (см. главы 31 и 32) являются наиболее сложными. Эти примеры посвящены средствам `.NET`, предназначенным для обработки наборов данных, включая их привязку (`binding`) к компонентам формы. В них также рассматривается XML-сериализация наборов данных, работа с подчиненными наборами данных и способы сортировки и поиска данных, основанные на применении функций сравнения и тестовых функций. Последний пример `HTOWERS` (см. главу 33) представляет собой компьютерную реализацию известной логической задачи "Ханойские башни". В этом примере рассматриваются особенности, связанные с созданием компонентов в ходе выполнения программы. Помимо обычного режима, в котором пользователь может самостоятельно решать задачу (используя перетаскивание `drag & drop`), приложение имеет демонстрационный режим, показывающий правильную последовательность действий для решения задачи. В заключительном разделе примера `HTOWERS` описывается, как задать идентификационные данные для разработанного приложения.

На прилагаемом компакт-диске содержатся все проекты, описанные в книге, причем для каждого проекта приводится несколько вариантов, соответствующих различным этапам его разработки.

Книга написана при поддержке внутреннего гранта К-07-Т-87 Южного федерального университета.



ЧАСТЬ I



Глава 1

Разработка программ в среде Microsoft Visual Studio .NET

В книге описываются приемы работы в двух вариантах интегрированных сред, созданных корпорацией Microsoft и ориентированных на платформу .NET Framework. Первый вариант — свободно распространяемая бесплатная система Microsoft Visual C# Express Edition, второй вариант — коммерческая система Microsoft Visual Studio, позволяющая разрабатывать приложения на различных языках платформы .NET. В среде Microsoft Visual Studio предусмотрены средства для разработки существенно большего числа типов приложений, однако в отношении стандартных Windows-приложений возможности данной среды по существу совпадают с возможностями ее экспресс-варианта для языка C#.

В настоящее время наиболее распространены две версии систем Microsoft Visual C# Express Edition и Microsoft Visual Studio: версия 2005, ориентированная на платформу .NET 2.0, и новая версия 2008, ориентированная на платформу .NET 3.5. В книге рассматриваются возможности, имеющиеся в обеих версиях.

В дальнейшем под средой Visual C# мы будем подразумевать как среду Microsoft Visual C# 2005/2008, входящую в состав системы Microsoft Visual Studio 2005/2008, так и ее экспресс-вариант.

1.1. Создание, сохранение и открытие проекта

При запуске среды Visual C# в нее автоматически загружается стартовая страница (**Start Page**), позволяющая быстро загрузить один из ранее разрабатывавшихся проектов (список **Recent Projects**), открыть какой-либо другой существующий проект (пункт **Open: Project...**) и создать новый проект (пункт **Create: Project...**). Впрочем, все отмеченные возможности доступны также из меню среды Visual C#:

- ❑ **File | New | Project...** или комбинация клавиш <Ctrl>+<Shift>+<N> — создание нового проекта (в экспресс-варианте данная команда является командой меню второго уровня: **File | New Project...**);
- ❑ **File | Open | Project/Solution...** или комбинация клавиш <Ctrl>+<Shift>+<O> — открытие существующего проекта (в экспресс-варианте команда имеет вид **File | Open Project...**);
- ❑ **File | Recent Projects** — загрузка одного из недавно разрабатывавшихся проектов.

Среда Visual C# организована таким образом, что в ней нельзя создать "отдельно взятый" проект. Каждый проект должен содержаться в особой сущности, называемой solution ("решение"), которую можно охарактеризовать как *группу взаимосвязанных проектов* (по этой причине в данной книге термин solution мы будем переводить как "группа проектов"). В каждый момент времени в среде Visual C# можно загрузить только одну группу проектов; загрузка другой группы приводит к автоматическому закрытию предыдущей.

При создании нового проекта на экране возникает диалоговое окно, в котором надо выбрать тип проекта и его имя. В качестве типа для всех проектов, рассматриваемых в книге (кроме двух первых проектов DISKINFO и EXCEP), следует выбирать **Windows Application**; в качестве имени рекомендуется указывать имя примера, приведенное в заголовке соответствующей главы книги, например, EVENTS (*см. главу 4*). Заметим, что имя проекта может содержать не только цифры и латинские буквы, но и другие символы, допустимые в именах файлов, в том числе пробелы и русские буквы, хотя этой возможностью не следует злоупотреблять.

В среде Visual Studio при создании нового проекта необходимо сразу указать каталог для его сохранения (поле **Location**). На размещение проекта также влияет информация, связанная с группой проектов (solution), в которую будет помещен создаваемый проект, в частности, флажок **Create directory for solution** (Создать каталог для группы проектов). Если в группу будет входить единственный проект, то следует снять данный флажок; это приведет к тому, что созданная группа проектов будет иметь то же имя, что и созданный проект, а в каталоге, указанном в поле **Location**, будет создан каталог с именем проекта, в котором будут размещаться все файлы, связанные с проектом и его группой.

При установленном флажке **Create directory for solution** можно указать имя группы проектов, отличное от имени самого проекта. В этой ситуации создается более сложная иерархия каталогов: в каталоге, указанном в поле **Location**, будет создан каталог с именем группы проектов, а в нем — каталог с именем проекта.

Отметим еще одну возможность среды Visual Studio (которой мы, однако, пользоваться не будем): создаваемый проект можно добавить к текущей группе проектов, т. е. группе, ранее загруженной в среду Visual Studio.

В экспресс-варианте Visual C# при создании нового проекта не требуется указывать его расположение; вся необходимая информация запрашивается при первом *сохранении* созданного проекта, поэтому рекомендуется выполнить это сохранение сразу после создания проекта.

Для сохранения всех изменений, внесенных в текущий проект (а точнее, в текущую группу проектов), в Visual C# предусмотрена команда **File | Save All**, а также клавиатурная комбинация <Ctrl>+<Shift>+<S>.

При открытии существующего проекта на экране появляется диалоговое окно **Open**, в котором можно выбрать либо файл с расширением sln (содержащий информацию о группе проектов с указанным именем), либо файл с расширением csproj (содержащий информацию о проекте с указанным именем). Однако даже при выборе отдельного проекта загружается группа проектов, в которой он содержится.

Некоторые примеры, описанные в книге, предполагают не создание нового проекта "с нуля", а модификацию уже имеющегося проекта. Имена таких примеров снабжены порядковыми номерами, начиная с номера 2 (например, PNGEDIT2—PNGEDIT4). Перед началом выполнения подобных примеров следует скопировать в новый каталог все файлы проекта из примера с предыдущим номером, после чего загрузить полученную копию проекта в среду Visual C# и начать ее модификацию.

Действия по модификации проекта опишем на примере преобразования проекта PNGEDIT1 в проект PNGEDIT2.

Вначале следует изменить имя проекта и имя связанной с ним группы проектов. Это проще всего выполнить с помощью окна **Solution Explorer**, обычно расположенного в правой части экрана (рис. 1.1): достаточно выделить в окне **Solution Explorer** строку, соответствующую проекту (на рис. 1.1 эта строка является выделенной) или группе проектов (данная строка начинается со слова Solution), нажать клавишу <F2>, ввести новое имя и нажать клавишу <Enter>.

Следует также изменить имя сборки (assembly name), т. е. имя результирующего exe-файла. Для этого надо выполнить команду **Project | <Имя проекта> Properties** (в нашем случае **Project | PNGEDIT2 Properties**), перейти в появившейся вкладке с именем проекта в раздел настроек **Application** и указать новое имя в поле **Assembly name** (рис. 1.2; для версии 2008 вид раздела **Application** несколько отличается от приведенного на рисунке). Название пространства

имен по умолчанию (поле **Default namespace**) изменять не следует, поскольку прежнее название используется во всех имеющихся cs-файлах проекта.

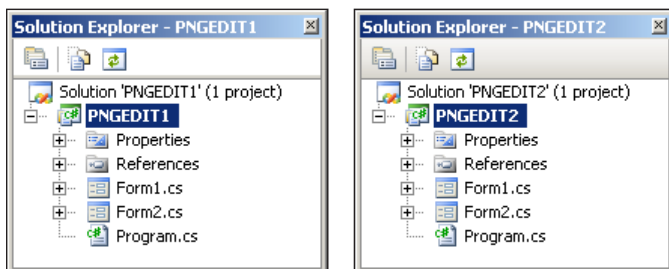


Рис. 1.1. Вид окна **Solution Explorer** до изменения имени проекта (*слева*) и после его изменения (*справа*)

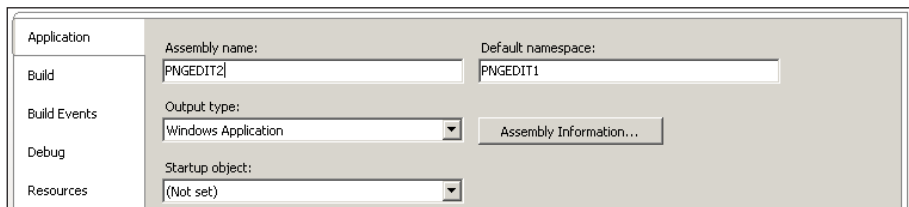


Рис. 1.2. Вид верхней части окна со свойствами проекта для версии Visual C# 2005

1.2. Добавление к проекту новой формы и размещение в форме нового компонента

При создании нового проекта типа Windows Application связанная с ним главная форма (класс с именем `Form1`) создается автоматически и сразу загружается в среду Visual C#. С формой `Form1` связывается набор файлов, основными из которых являются файлы `Form1.cs` и `Form1.Designer.cs`. В этих файлах содержится описание данной формы на языке C#, причем второй из них содержит ту часть описания, которая генерируется автоматически в ответ на различные действия разработчика, связанные с визуальным проектированием (собственный код разработчик обычно размещает в файле `Form1.cs`).

Если в проекте требуется использовать дополнительные формы (*см., например, главу 5*), то их проще всего добавить в проект с помощью команды меню **Project | Add Windows Form...** При выполнении этой команды возникает диалоговое окно, в котором надо указать имя добавляемой формы (которое одновременно будет и начальной частью имен файлов, содержащих ее описание).

В книге всегда используются имена форм, предлагаемые системой по умолчанию (Form1, Form2 и т. д.).

Любая созданная форма отображается в редакторе Visual C# в *режиме дизайна* (на экране появляется изображение формы и ее содержимого); соответствующая вкладка редактора заканчивается текстом **[Design]**. Например, для главной формы Form1 вкладка содержит текст **Form1.cs [Design]**. Для перехода к содержимому соответствующего cs-файла достаточно нажать клавишу <F7>; при этом в редакторе появится новая вкладка с текстом данного файла (ярлычок этой вкладки будет содержать имя файла, например, **Form1.cs**). Для обратного переключения с текста на изображение формы можно использовать комбинацию клавиш <Shift>+<F7>. Переключаться между изображением формы и текстом кода можно также с помощью *контекстного меню*: при щелчке правой кнопкой мыши на изображении формы первый пункт появляющегося меню имеет имя **View Code** и позволяет перейти к тексту cs-файла, а при вызове контекстного меню для текста cs-файла первый пункт меню имеет имя **View Designer** и позволяет вернуться в режим дизайна.

Другим удобным способом переключения между различными окнами среды Visual C# является использование комбинации клавиш <Ctrl>+<Tab>: после ее нажатия на экране появляется вспомогательное окно со списком всех загруженных файлов и форм (Active Files), а также всех открытых вспомогательных окон (Active Tool Windows). На рис. 1.3 приведен примерный вид вспомогательного окна для версии 2005 (в версии 2008 окно имеет незначительные отличия). После появления вспомогательного окна надо, не отпуская клавиши <Ctrl>, выделить имя файла или окна, которое надо активизировать (для выбора можно использовать клавиши со стрелками и клавишу <Tab>), а затем отпустить клавишу <Ctrl>.

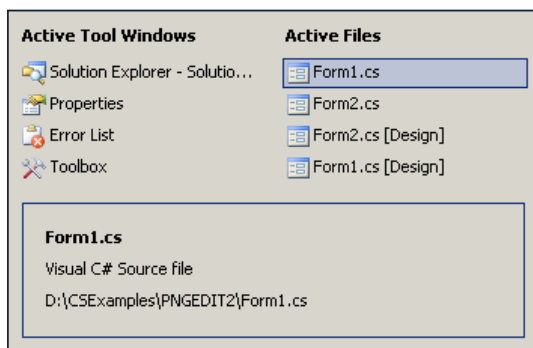



Рис. 1.3. Вспомогательное окно для выбора одного из загруженных файлов или окон среды Visual C#

Любую вкладку в редакторе можно закрыть. Для этого достаточно сделать ее активной и нажать комбинацию клавиш <Ctrl>+<F4> или вызвать контекстное меню вкладки, щелкнув правой кнопкой мыши на ее ярлычке, и выбрать пункт **Close**.

Если после загрузки существующего проекта в редактор не загрузилась требуемая форма, достаточно выполнить двойной щелчок мышью на имени этой формы в окне **Solution Explorer** (см. рис. 1.1). Можно также щелкнуть на имени этой формы *правой* кнопкой мыши и выбрать пункт **Open** из появившегося контекстного меню. Данное меню также содержит пункт **View Code**, позволяющий загрузить в редактор текст cs-файла, и пункт **View Designer**, позволяющий загрузить в редактор форму в режиме дизайнера. При выборе одной из форм в окне проекта **Solution Explorer** дополнительно появляются кнопки быстрого доступа , первая из которых позволяет загрузить текст файла, а вторая — отобразить форму в режиме дизайнера.

Для размещения в форме нового компонента следует воспользоваться *окном компонентов Toolbox* (отобразить это окно на экране можно либо командой меню **View | Toolbox**, либо комбинацией клавиш <Ctrl>+<Alt>+<X>, либо кнопкой ). Это окно обычно отображается в левой части экрана; на рис. 1.4 приведен вид верхней части окна **Toolbox** при выборе в нем группы **All Windows Forms**. Заметим, что компоненты отображаются в окне **Toolbox** только в том случае, если редактор находится в режиме дизайнера. Надо выбрать в этом окне группу, содержащую нужный компонент (щелкнув мышью на имени этой группы), затем щелкнуть на изображении нужного компонента, выделив его (на рис. 1.4 выделен компонент `Button`), и, наконец, щелкнуть в том месте формы, где предполагается разместить выбранный компонент.

Для того чтобы выбрать компонент в окне **Toolbox**, необязательно знать, в какой группе он содержится, поскольку в этом окне имеется группа **All Windows Forms**, в которой указаны все имеющиеся компоненты в алфавитном порядке (рис. 1.4).

Если в окне **Toolbox** отсутствует нужный компонент, то можно попытаться загрузить его в это окно. Действия, которые при этом надо выполнить, описываются в *разд. 13.1*.

Для быстрого размещения в форме нескольких компонентов одного типа следует после выделения требуемого компонента в окне **Toolbox** выполнить щелчок мышью на форме несколько раз, держа нажатой клавишу <Ctrl>. Если в окне **Toolbox** требуется отменить выбор компонента, не размещая его в форме, то достаточно выделить в этом окне элемент **Pointer** (он располагается первым в любой группе компонентов; рядом с ним изображен курсор в виде стрелки — см. рис. 1.4).

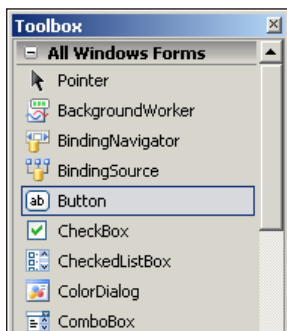


Рис. 1.4. Вид верхней части окна **Toolbox**

Если компонент требуется разместить не в форме, а в другом компоненте (например, в компоненте `Panel`), то при размещении компонента необходимо выполнить щелчок мыши в области компонента-приемника. В качестве компонентов-приемников могут использоваться только особые компоненты, называемые *компонентами-контейнерами* (сама форма также является компонентом-контейнером). Компонент-контейнер (форма, панель и т. д.), содержащий другие компоненты, называется *родительским* (*parent*) по отношению к размещенным в нем *дочерним* компонентам.

При описании этапов разработки проекта мы всегда будем указывать, в каком компоненте-контейнере следует разместить каждый компонент.

После добавления компонента в форму он автоматически получает *имя*, которое сохраняется в его свойстве `Name`. По умолчанию имя любого компонента начинается со строчной буквы и состоит из имени типа компонента и порядкового номера. Так, первая размещенная в форме кнопка (компонент типа `Button`) получит имя `button1`, а вторая кнопка — имя `button2`. Формы имеют имена, начинающиеся с прописной буквы (`Form1`, `Form2` и т. д.); это связано с тем, что данные имена являются именами новых *классов* — потомков базового класса `Form`. Имя, присвоенное по умолчанию, всегда можно заменить на другое имя, позволяющее, например, уточнить назначение того или иного компонента (например, кнопку `button1`, содержащую текст **ОК**, можно назвать `btnOK`). Однако в проектах, описываемых в данной книге, почти всегда используются имена компонентов, предлагаемые системой Visual C# по умолчанию: это позволяет уменьшить количество действий по настройке свойств компонентов и упрощает ориентировку в текстах программ. "Значимые" имена используются только для пунктов меню, кнопок быстрого доступа и разделов статусной панели (см. главы 18, 21, 22). Следует заметить, однако, что при разработке больших проектов целесообразно использовать значимые имена для всех компонентов.


При описании действий по добавлению компонента в форму почти никогда не уточняется, как именно *позиционировать* компонент на его родительском компоненте, поскольку это легко определить по приводимому рисунку. Перечислим способы позиционирования компонента:

- перетаскивание мышью по форме (при этом на форме появляются вспомогательные *линии выравнивания*, позволяющие осуществить привязку компонента к границам формы или разместить его на уровне границ существующих компонентов);
- использование панели выравнивания **Layout** (см. разд. 9.1);
- перемещение на один пиксел с помощью клавиш со стрелками;
- явное указание значения свойства `Location` в окне **Properties** (работа с окном **Properties** подробно рассматривается в разд. 1.3).

Размеры визуальных компонентов также обычно не уточняются. Для настройки размеров можно воспользоваться перетаскиванием мышью за один из маркеров, окружающих выделенный компонент, или клавишами со стрелками при нажатой клавише `<Shift>`. Можно также явно задать значения свойства `Size` в окне **Properties** или воспользоваться панелью **Layout**.

Дополнительные сведения, связанные с настройкой меню приложения, его панели инструментов и статусной панели, приводятся в *главах 18, 21 и 22*.

1.3. Настройка свойств форм и компонентов

Для настройки свойств форм и компонентов используется *окно свойств Properties*, быстро перейти в которое можно с помощью комбинации клавиш `<Alt>+<Enter>`. Окно **Properties** обычно располагается в правом нижнем углу экрана и может находиться в двух режимах: **Properties** и **Events** (рис. 1.5). В настоящем разделе мы рассмотрим режим **Properties**, предназначенный для отображения *свойств* выделенного компонента или группы компонентов (для перехода в данный режим надо нажать третью кнопку  на панели инструментов окна свойств).

Если выделена группа компонентов, то в окне свойств отображаются только те свойства, которые имеются у всех выделенных компонентов.

Выделение группы компонентов позволяет быстро задать для них одинаковые размеры, заголовки или другие общие свойства, а также переместить их, сохраняя взаимное расположение ("как одно целое"). Перечислим способы выделения группы компонентов:

- щелчок на компонентах при нажатой клавише `<Shift>` или `<Ctrl>`;

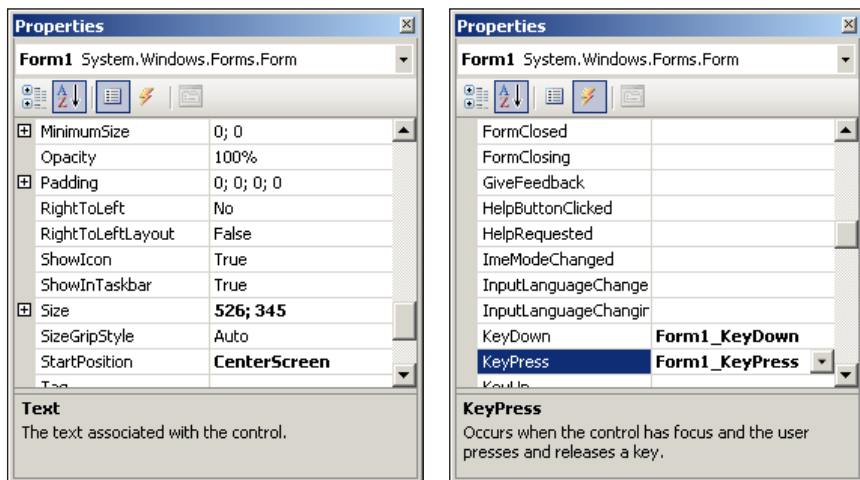


Рис. 1.5. Окно **Properties** в режимах **Properties** (слева) и **Events** (справа)

- охват компонентов пунктирной рамкой, которая появляется на форме при перемещении мыши с нажатой левой кнопкой (для выделения достаточно захватить рамкой часть компонента).

Если выделена группа компонентов, то один компонент этой группы является *текущим* (его маркеры имеют белый цвет). На рис. 1.6 приведен фрагмент формы с четырьмя выделенными компонентами; текущим является компонент `button2`. Результат некоторых действий (например, связанных с выравниванием компонентов на форме — см. разд. 9.1) зависит от того, какой компонент выделенной группы является текущим. Для того чтобы сделать текущим другой компонент из выделенной группы, достаточно выполнить на нем щелчок мышью. Для снятия выделения с группы компонентов надо выделить компонент, не входящий в эту группу.

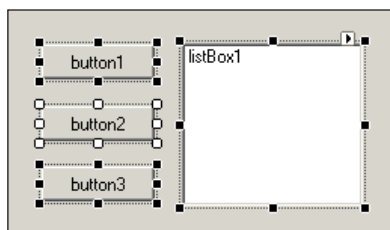



Рис. 1.6. Фрагмент формы с группой выделенных компонентов

Полезно знать, что если выделен некоторый компонент, то для выделения его родительского компонента достаточно нажать клавишу `<Esc>` (таким способом можно быстро выделить любой компонент-контейнер, даже если его целиком

покрывают дочерние компоненты). Для выделения формы есть еще более простой способ: щелчок мышью на ее *заголовке*.

С помощью двух первых кнопок окна свойств  (см. рис. 1.5) можно настроить способ отображения свойств: по категориям (первая кнопка) или в алфавитном порядке имен (вторая кнопка). Каждая категория имеет имя (например, **Appearance**) и содержит "родственные" свойства. Используя группировку свойств по алфавиту, проще перейти к требуемому свойству; в то же время, при начальном ознакомлении со свойствами, имеющимися у компонента, удобнее использовать группировку по категориям.

Настройку свойств форм и компонентов мы будем описывать в листингах, озаглавленных "Настройка свойств". Примером такого листинга является листинг 1.1.

В листингах настройки свойств вначале указывается имя компонента, свойства которого требуется изменить, а затем, после двоеточия, — список его настраиваемых свойств в формате

```
ИМЯ СВОЙСТВА = новое значение свойства
```

(значение свойства выделяется полужирным шрифтом). Свойства перечисляются через запятую.

Компоненты могут иметь *составные свойства*, т. е. свойства, имеющие собственные свойства (например, `Font`). Если требуется настроить одно или несколько свойств составного свойства, в листинге настройки свойств используется разделитель "точка" (соответствующие примеры приводятся в листинге 1.2). Составные свойства помечаются в окне **Properties** знаком "+" слева от имени (на рис. 1.5 знаком "+" помечены свойства `minimumSize`, `padding` и `size`). Щелчок на знаке "+" приводит к отображению в окне всех свойств выбранного составного свойства; знак "+" при этом меняется на "-". Щелчок на знаке "-" сворачивает список свойств составного свойства.

Листинг 1.2 демонстрирует еще одно используемое обозначение: если требуется очистить какое-либо свойство, то в качестве его значения указывается курсивный текст *"пустая строка"*.


Для значений логических свойств в листингах "Настройка свойств" используются константы **True** и **False**, начинающиеся с заглавной буквы (в отличие от ключевых слов языка C# `true` и `false`); это связано с тем, что именно так называются логические константы в окне **Properties**. Обозначения **True** и **False** будут также использоваться в тексте примеров при описании действий, связанных с окном **Properties**. Однако при комментировании фрагментов *программного кода* мы будем обозначать логические константы так, как это принято в языке C#: `true` и `false`.


Листинг 1.1. Настройка свойств (фрагмент листинга 5.1)

```
Form1: Text = Главное окно, MaximizeBox = False,  
    FormBorderStyle = FixedSingle  
button1: Text = Открыть подчиненное окно  
button2: Text = Открыть диалоговое окно
```


Листинг 1.2. Настройка свойств (копия листинга 30.3)

```
Form2: Text = пустая строка, ControlBox = False,  
    FormBorderStyle = FixedSingle, Opacity = 80%, ShowInTaskbar = False,  
    StartPosition = CenterScreen, UseWaitCursor = True  
label1: Text = Тригонометрические функции, AutoSize = False, Dock = Fill,  
    TextAlign = MiddleCenter, Font.Name = Times New Roman, Font.Size = 32,  
    Font.Bold = True
```

Для задания некоторых свойств, связанных с привязкой или выравниванием (например, `Dock` или `TextAlign`), в окне **Properties** используются *вспомогательные панели*. Для отображения таких панелей предназначена кнопка , появляющаяся при выделении соответствующего свойства. В этих панелях надо выбрать один или несколько элементов, расположенных в требуемой позиции (слева, справа, по центру и т. п.), причем результат выбора будет отображен в окне **Properties** в виде обычного текста (например, `Bottom` или `MiddleCenter`). Хотя подобные действия являются интуитивно понятными, они, как правило, снабжаются в тексте примеров дополнительными пояснениями.

В некоторых случаях для задания свойства бывает удобно воспользоваться специальным *диалоговым окном*. Если для свойства предусмотрено такое окно, то при выделении свойства в окне **Properties** справа от него изображается кнопка с многоточием , позволяющая вызвать диалоговое окно (примером такого свойства является `Font`).

1.4. Определение обработчиков событий

У всех компонентов имеется не только набор свойств, но и набор *событий*, с которыми можно связать методы — *обработчики событий*. Список обработчиков для выделенного компонента или группы компонентов отображается в окне свойств в режиме **Events** (см. рис. 1.5); для перехода в данный режим надо нажать четвертую кнопку  на панели инструментов окна **Properties**.

Список событий, как и список свойств, можно упорядочивать двумя способами: по категориям и по именам (в алфавитном порядке).


Двойной щелчок мышью на *пустом* поле ввода рядом с именем нужного события обеспечивает автоматическое создание заготовки для обработчика этого события. Обработчик любого события для любого компонента оформляется как метод той формы, на которой расположен данный компонент. Во всех примерах книги используются имена методов-обработчиков, предлагаемые системой Visual C# по умолчанию: это позволяет легко определить, с каким компонентом и каким событием связан обработчик.

Тексты обработчиков приводятся в книге в листингах, заголовок которых начинается со слова "Обработчик" (см. листинг 1.3, являющийся копией листинга 4.3).

Листинг 1.3. Обработчик Form1.MouseDown

```
private void Form1_MouseDown(object sender, MouseEventArgs e)
{
    button1.Location = new Point(e.X - button1.Width / 2,
        e.Y - button1.Height / 2);
}
```

Полужирным шрифтом выделяются те строки, которые требуется добавить в автоматически созданную заготовку метода-обработчика. Иногда фрагменты программы снабжаются комментариями, хотя чаще пояснения даются в основном тексте примера или в разделе "Комментарии".

Если в качестве обработчика нужно указать имя *уже имеющегося* метода, то достаточно выбрать имя этого метода в выпадающем списке рядом с именем события в окне свойств (кнопка разворачивания списка  появляется только около выделенного события; на рис. 1.5 справа эта кнопка изображена рядом с событием `KeyPress`).

Так как при разработке программ в Visual C# необходимо иметь четкое представление о том, как создаются и используются обработчики событий, этой теме посвящен специальный проект EVENTS (см. главу 4). Связывание обработчика с несколькими событиями подробно обсуждается в главе 6.

1.5. Внесение изменений в текст программы

При выполнении примеров из данной книги изменения, как правило, вносятся в текст уже имеющихся обработчиков. Если изменения являются существенными и охватывают весь текст обработчика, то приводится его новый полный

текст, в котором измененные или добавленные строки или фрагменты строк выделяются полужирным шрифтом (см. листинг 1.4, являющийся копией листинга 6.7).

Листинг 1.4. Новый вариант метода `button1_Click`

```
private void button1_Click(object sender, EventArgs e)
{
    label1.Text = (sender as Button).Text[1].ToString();
    label2.Text = "=";
}
```

Если же изменения незначительны, а обработчик достаточно велик, то просто указывается, какие операторы надо добавить или заменить. Приведем два примера подобных исправлений.

В метод `SaveToFile` формы `Form1` добавьте оператор

```
textBox1.Modified = false;
```

В методе `button2_Click` формы `Form1` вставьте после оператора

```
Image im = new Bitmap(s);
```

два следующих оператора:

```
Graphics g = Graphics.FromImage(im);
```

```
g.Dispose();
```

Если место добавления не уточняется, то операторы должны добавляться *в конец* указанного метода.

Перейти к уже имеющемуся обработчику для его корректировки можно, либо перемещаясь по тексту файла, либо с помощью окна свойств, выполнив двойной щелчок мышью на поле ввода с именем нужного обработчика.

Аналогичным образом описываются изменения текста программы, не связанные с конкретным обработчиком. Приведем два примера.

В описание класса `Form1` добавьте поле

```
private Form2 form2 = new Form2();
```

В конструктор класса `Form1` добавьте оператор


```
AddOwnedForm(form2);
```

ПРИМЕЧАНИЕ

В качестве русского эквивалента английского термина `operator` в книге используется термин "операция", например, "операция +", в то время как термин "оператор" применяется для обозначения "программных инструкций" (statements).

При наборе и редактировании текста программ следует использовать дополнительные возможности редактора Visual C#. Опишем некоторые из них.

- Если ввести имя объекта (например, `button1`) и точку после него, то на экране появится список всех методов и свойств, которые имеет данный объект, причем для быстрого перехода к нужному методу достаточно набрать несколько его первых символов. Для вставки в текст программы имени выбранного метода или свойства надо нажать клавиши `<Enter>`, `<Tab>` или `<Пробел>`. Список методов и свойств можно вызвать и явным образом, нажав комбинацию клавиш `<Ctrl>+<Пробел>`.
- После ввода имени метода и скобки (на экране появляется подсказка с кратким описанием этого метода и списком всех его параметров. Если метод является *перегруженным*, т. е. может вызываться с различным набором параметров, то можно просмотреть все его перегруженные варианты, нажимая клавиши `<↑>` и `<↓>`. Для вызова подобной подсказки можно также нажать комбинацию клавиш `<Ctrl>+<Shift>+<Пробел>`.
- Редактор среды Visual C# позволяет автоматически *форматировать* текст программы, выполняя правильную расстановку *отступов* и *пробелов* в каждой строке кода. В частности, форматирование операторов, заключенных в блок `{ }`, выполняется при вводе фигурной скобки `}`, завершающей этот блок. Кроме того, предусмотрена команда, позволяющая отформатировать весь текст, содержащийся в файле; для этого достаточно последовательно нажать две комбинации клавиш: `<Ctrl>+<K>` и `<Ctrl>+<D>`. Следует заметить, что форматирование выполняется только для синтаксически правильного программного кода, т. е. кода, в котором не выделен как ошибочный ни один из его фрагментов (ошибочные фрагменты кода подчеркиваются красной волнистой линией). Во всех листингах, приводимых в книге, сохранено форматирование, выполненное редактором среды Visual C#; изменен лишь *размер отступа*: вместо 4 пробелов, установленных в редакторе по умолчанию, используется отступ, равный 2 пробелам (это позволяет разместить в одной строке листинга операторы, которые при стандартной величине отступа потребовалось бы разбивать на две строки).
- Для быстрого перехода к нужному фрагменту программы удобно использовать *закладки* (bookmarks). Для установки/отмены закладки на текущей строке программы (т. е. строке, содержащей клавиатурный курсор) надо нажать комбинацию клавиш `<Ctrl>+` и затем клавишу `<T>`, а для перехода

к следующей или предыдущей установленной закладке надо нажать комбинацию клавиш <Ctrl>+ и затем клавишу <N> или <P> соответственно. Можно также использовать меню **Edit | Bookmarks** и кнопки быстрого доступа  на панели **Text Editor**.


- Если требуется *закомментировать* какой-либо фрагмент программы, то достаточно выделить его и нажать комбинацию клавиш <Ctrl>+<E>, а затем клавишу <C>. Для того чтобы раскомментировать закомментированный фрагмент, надо выделить его и нажать комбинацию клавиш <Ctrl>+<E>, а затем клавишу <U>. Если требуется закомментировать или раскомментировать одну строку, то вместо ее выделения достаточно установить на ней клавиатурный курсор, после чего нажать указанные клавиши. Вместо клавиатурных комбинаций можно использовать кнопки быстрого доступа



на панели **Text Editor**.

- Редактор среды Visual C# обладает богатыми возможностями *поиска и замены*. Эти возможности подробно описываются в комментарии 2 к разд. 10.7.
- Наконец, следует упомянуть о возможности *автогенерации кода* с помощью специальных шаблонов (code snippets). Использование подобных шаблонов описывается в разд. 31.1 и комментарии к нему.

1.6. Запуск приложения

Каждый этап разработки проекта описывается в отдельном разделе главы, посвященной этому проекту. В любом разделе вначале перечисляются необходимые действия, связанные с модификацией проекта. Затем следует абзац, начинающийся со слова **Результат**. В этом абзаце описывается, как будет работать новый вариант программы. Появление абзаца **Результат** служит признаком того, что модифицированный проект можно откомпилировать и запустить на выполнение (для этого достаточно нажать клавишу <F5> или кнопку .

Если код программы был набран с синтаксическими ошибками, то сообщения об этих ошибках появляются в окне **Error List**, которое при этом становится активным. Для того чтобы перейти на строку программы, в которой обнаружена первая синтаксическая ошибка, достаточно нажать клавишу <↓> (в результате сообщение о первой ошибке будет подсвечено), после чего нажать клавишу <Enter>. Можно также выполнить двойной щелчок мышью на строке с сообщением об ошибке.

При компиляции программы в среде Visual C# важную роль играют две настройки среды, расположенные в группе **Projects and Solutions** окна **Options** (данное окно вызывается командой меню **Tools | Options...**). Первая настрой-

ка — флажок **Always show Error List if build finishes with errors** (Всегда отображать окно Error List, если компиляция завершилась с ошибками) в разделе **General**. Этот флажок должен быть установлен. Вторая настройка — выпадающий список **On Run, when build or deployment errors occur** (Если обнаружены ошибки компиляции или размещения при выполнении команды Run) в разделе **Build and Run**. В этом списке должен быть выбран вариант **Do not launch** (Не запускать программу). В экспресс-варианте среды Visual C# для отображения указанных настроек необходимо установить флажок **Show all settings** в окне **Options** (по умолчанию этот флажок не установлен).

Если после абзаца **Результат** следует абзац с пометкой **Ошибка**, значит, программа, несмотря на успешную компиляцию, будет работать не совсем правильно, и в нее надо внести дополнительные исправления (таким способом в примерах привлекается внимание к типичным ошибкам, которые могут возникнуть в аналогичных ситуациях). Если после абзаца **Результат** следует абзац с пометкой **Недочет**, значит, программа работает правильно и делает то, что требуется, но имеет дефекты интерфейса, т. е. ею неудобно пользоваться. Как правило, сразу после описания ошибки или недочета указывается способ их исправления, хотя иногда исправление откладывается до следующего раздела.

Глава 2



Консольное приложение: проект DISKINFO

Проект DISKINFO знакомит с приемами разработки консольных приложений. Описывается структура консольного приложения и подключаемые к нему пространства имен. Рассматривается форматированный вывод, в частности, использование управляющих последовательностей. Описываются классы `DriveInfo`, `StringBuilder` и `Environment`. Кроме того, описываются приемы работы с параметрами командной строки.

2.1. Создание консольного приложения

При создании нового проекта — консольного приложения следует выполнять практически те же действия, что и при создании Windows-приложения (см. разд. 1.1). Единственным отличием является указание другого типа проекта в окне **New Project**: для консольного приложения следует выбрать вариант **Console Application**.

Созданный проект будет содержать файл `Program.cs`, который сразу загрузится в редактор (листинг 2.1).

Листинг 2.1. Исходный текст файла `Program.cs` для консольного приложения

```
using System;
using System.Collections.Generic;
using System.Text;

namespace DISKINFO
{
    class Program
    {
```

```
static void Main(string[] args)
{
}
}
```

Первые три оператора в файле Program.cs содержат названия трех пространств имен, классы из которых используются наиболее часто (см. комментарий 1). Благодаря этим операторам, имена классов можно указывать в программе без уточнения, к какому пространству имен они принадлежат (например, вместо `System.Console` можно писать просто `Console`).

Метод `Main` является стартовой точкой выполнения программы. Его параметр `args` позволяет получить информацию о *параметрах командной строки*, указанных при запуске данной программы (использование параметра `args` демонстрируется в *разд. 2.3*).

Добавьте в метод `Main` новые операторы (листинг 2.2).

Листинг 2.2. Добавление к методу `Main`

```
Console.WriteLine("Программа DISKINFO\n");
Console.WriteLine("\nДля завершения программы нажмите <Enter>...");
Console.ReadLine();
```

Результат. При запуске программы на экране появляется особое *консольное окно*, используемое для ввода/вывода данных в текстовом режиме. Это окно обладает всеми свойствами окна приложения DOS; в частности, оно имеет расширенное системное меню, вызываемое комбинацией клавиш `<Alt>+<Пробел>`. При выполнении программы можно переключаться в полноэкранный режим комбинацией клавиш `<Alt>+<Enter>` (эта же комбинация восстанавливает оконный режим). Консольное окно будет содержать текст, приведенный в листинге 2.3.

После завершения программы консольное окно немедленно закрывается. Чтобы можно было ознакомиться с содержимым окна, в программу был добавлен вызов метода `ReadLine` класса `Console` (см. листинг 2.2, последний оператор). Этот метод предназначен для ввода строки, причем признаком завершения ввода является нажатие клавиши `<Enter>`. Поэтому до нажатия клавиши `<Enter>` консольное окно будет оставаться на экране. Строка, возвращаемая методом `ReadLine`, в программе не используется, поэтому сохранять ее в какой-либо переменной нет необходимости.

Листинг 2.3. Содержимое консольного окна при выполнении программы

Программа DISKINFO

Для завершения программы нажмите <Enter>...

ПРИМЕЧАНИЕ

Следует отметить, что в консольном окне русский текст будет отображаться правильно, несмотря на то, что в этом окне используется особая DOS-кодировка OEM 866 (MS-DOS Russian), в которой коды русских букв отличаются от кодов в стандартной Windows-кодировке ANSI 1251 (Windows Cyrillic). Перекодирование текста при выполнении консольных приложений в .NET выполняется автоматически, при этом учитываются языковые настройки Windows (в нашем случае — настройки для русского языка).

Комментарии

1. Дадим краткую характеристику пространствам имен, автоматически подключенным к нашему приложению (см. листинг 2.1, первые три оператора). Пространство имен `System` является основным пространством стандартной библиотеки .NET Framework; оно содержит определения классов, необходимых практически в любой программе. В частности, в нем определен класс `Console`, обеспечивающий ввод/вывод для консольного окна. Пространство имен `System.Collections.Generic` содержит *обобщенные* классы-коллекции. Классы-обобщения (generics) появились в версии .NET 2.0. Способ реализации коллекций, основанный на обобщениях, позволяет при описании коллекции указать, данные какого типа будут в ней содержаться (пример использования обобщенного класса-коллекции приводится в *разд. 7.1*). Пространство имен `System.Text` содержит различные классы и перечисления, предназначенные для обработки строковых данных; в *разд. 2.2* будет использован класс `StringBuilder`, определенный в этом пространстве имен.
2. Комбинация `\n`, использованная в строковых константах (см. листинг 2.2), является одной из *управляющих последовательностей* (*escape-последовательностей*), которые можно указывать в строковых выражениях. Эта комбинация обозначает символ с кодом 10 (переход на новую строку). Среди других управляющих последовательностей отметим `\\` (символ `\`), `\"` (двойная кавычка), `\'` (одинарная кавычка), `\0` (символ с кодом 0), `\r` (возврат каретки — символ с кодом 13), `\b` (символ с кодом 8, генерируемый клавишей <Backspace>), `\t` (табуляция — символ с кодом 9) и `\uN` (символ

Unicode с шестнадцатеричным кодом n). В управляющей последовательности `\uN` код n должен состоять из 4 цифр и может иметь значение от 0000 до FFFF; например, символ `\u0041` обозначает латинскую букву А (десятичный код 65).

3. Если запустить консольное приложение не в режиме **Debug** (клавишей `<F5>`), а в режиме **Release** (комбинацией клавиш `<Ctrl>+<F5>`), то при его завершении в консольном окне появится текст "Для продолжения нажмите любую клавишу..." и окно будет закрыто только после нажатия произвольной клавиши. Таким образом, в этом режиме нет необходимости в операторе `Console.ReadLine` (более того, при наличии этого оператора для закрытия приложения в режиме **Release** придется вначале нажать клавишу `<Enter>`, а затем еще одну произвольную клавишу). Однако при запуске полученного exe-файла непосредственно из Windows (а не из среды Visual C#) приложение выполняется так же, как в режиме **Debug**, т. е. сообщение "Для продолжения нажмите любую клавишу..." не возникает.

2.2. Получение информации о текущем диске

В начало файла `Program.cs` добавьте оператор

```
using System.IO;
```

В класс `Program` добавьте новый метод `DInfo` (листинг 2.4) и измените метод `Main` (листинг 2.5).

Листинг 2.4. Метод `DInfo` класса `Program`

```
static void DInfo(string path)
{
    string none = "---",
        d = path[0].ToString().ToUpper();
    DriveInfo di = new DriveInfo(d);
    StringBuilder s = new StringBuilder(40);
    s.AppendFormat("{0,-4}", d);
    if (di.DriveType != DriveType.NoRootDirectory)
    {
        s.AppendFormat("{0,-9}", di.DriveType);
        if (di.IsReady)
            s.AppendFormat("{0,12:N0} {1,12:N0}", di.TotalSize / 1024,
```

```

        di.TotalFreeSpace / 1024);
    else
        s.AppendFormat("{0,12} {0,12}", none);
    }
    else
        s.AppendFormat(" {0,-9}{0,12} {0,12}", none);
    Console.WriteLine(s);
}

```

Листинг 2.5. Новый вариант метода Main

```

static void Main(string[] args)
{
    Console.WriteLine("Программа DISKINFO\n");
    Console.WriteLine(" Disk Type          Size (K)      Free (K)");
    Console.WriteLine(new String(' ', 40));
    DInfo(Environment.CurrentDirectory);
    Console.WriteLine("\nДля завершения программы нажмите <Enter>...");
    Console.ReadLine();
}

```

Результат. При запуске программы в консольном окне отображается информация о текущем диске (листинг 2.6).

Листинг 2.6. Содержимое консольного окна при выполнении программы

Программа DISKINFO

```

Disk Type          Size (K)      Free (K)
=====
D   Fixed          16 370 256    4 292 728

```

Для завершения программы нажмите <Enter>...

Комментарии

1. Метод `DInfo` в качестве параметра принимает строку `path`, из которой используется только первый символ (предполагается, что данный символ является *буквой диска*). Поскольку выражение `path[0]` имеет тип `char`, для

возможности использования этого выражения в качестве строкового параметра конструктора класса `DriveInfo` необходимо выполнить его явное преобразование к типу `string` с помощью метода `ToString`. Для повышения наглядности полученная односимвольная строка затем преобразуется к верхнему регистру с помощью метода `ToUpper` класса `string`.

- Для получения информации о диске используется класс `DriveInfo`, появившийся в версии .NET 2.0 и определенный в пространстве имен `System.IO`. Для создания объекта класса `DriveInfo` достаточно вызвать его конструктор, указав в нем букву требуемого диска. Свойство `DriveType` (типа перечисления `DriveType`) позволяет определить тип дискового носителя, причем даже в случае, если диск недоступен (среди возможных значений перечисления `DriveType` укажем `Removable`, `Fixed`, `Network` и `CDRom`). Для доступного диска можно определить его размер в байтах (свойство `TotalSize` типа `long`), размер свободного пространства в байтах (свойство `TotalFreeSize` типа `long`) и его метку (строковое свойство `VolumeLabel`, которое доступно и для чтения, и для записи).

Заметим, что тип `long`, используемый для свойств `TotalSize` и `TotalFreeSize`, позволяет хранить числа в диапазоне от $-9\,223\,372\,036\,854\,775\,808$ до $9\,223\,372\,036\,854\,775\,807$ (таким образом, имеется принципиальная возможность определять размер диска, содержащего более 9 млн терабайтов).

- Для формирования строки с информацией о диске используется объект `s` класса `StringBuilder`. Этот класс определен в пространстве имен `System.Text` и позволяет более эффективно (по сравнению с классом `string`) выполнять операции, связанные с преобразованием строк. В конструкторе класса `StringBuilder` мы указали *емкость* формируемой строки (т. е. ее максимально возможный размер, равный 40 символам), однако это не означает, что класс `s` не может содержать строки большего размера: если размер фактической строки, помещаемой в объект типа `StringBuilder`, превышает его емкость, то происходит автоматическое удвоение емкости. Явное указание емкости в конструкторе лишь позволяет избежать лишних операций по выделению и освобождению памяти.

Важной особенностью объектов типа `StringBuilder`, по сравнению с объектами типа `string`, является доступность их символов не только на чтение, но и на запись. Кроме того, любые действия по изменению строки типа `StringBuilder` выполняются над этой же строкой и не приводят к появлению новой (измененной) строки, как это происходит в методах класса `string`, связанных с изменением строковых данных. Используемый в программе метод `AppendFormat` добавляет к прежнему содержимому строки новые данные, предварительно выполняя их форматирование.

4. При форматировании данных в методе `AppendFormat` используются специальные *форматные настройки*, имеющие в общем случае вид `{ind, width :spec}`, где `ind` определяет индекс формируемого элемента в последующем списке параметров (индексирование проводится от 0), а `width` определяет минимальную ширину поля вывода для формируемого элемента и способ его выравнивания в пределах данного поля: если значение `width` положительно, то выполняется выравнивание по правой границе поля вывода, а если отрицательно — то по левой. Атрибут `spec` задает *спецификатор формата* для данного элемента. Используемый в программе спецификатор `n` позволяет выводить число с пробелами — разделителями тысяч, а указанный после него `0` означает нулевое количество дробных знаков (по умолчанию в формате `n` выводятся два дробных знака). Из других спецификаторов формата отметим `c` (денежный формат), `d` (десятичный целочисленный формат), `x` (шестнадцатеричный целочисленный формат), `e` (экспоненциальный числовой формат), `f` (числовой формат с фиксированной точкой), `p` (процентный формат). Спецификаторы формата можно использовать во многих методах, связанных с формированием и выводом строк, например, в методе `Format` класса `string` и методах `Write` и `WriteLine` класса `Console`. Из атрибутов, входящих в форматные настройки, только атрибут `ind` является обязательным; при этом допустимо указывать несколько *одинаковых* значений атрибута `ind`, если требуется вывести один и тот же элемент данных несколько раз (см. два последних вызова метода `AppendFormat` в листинге 2.4). Если отсутствует атрибут `width`, то используется ширина поля вывода, минимально необходимая для отображения отформатированного элемента. Если отсутствует атрибут `spec`, то выбирается вариант форматирования по умолчанию (этот вариант соответствует спецификатору формата `g`). При отсутствии `width` предшествующая запятая в форматных настройках не указывается; при отсутствии `spec` не указывается предшествующее двоеточие.
5. Вместо явного задания строки, содержащей 40 символов = (знак равенства), мы воспользовались вариантом конструктора `string(c, n)`, позволяющим создать строку указанной длины `n`, состоящую из одинаковых символов `c`.
6. Для определения текущего диска использовалось свойство `CurrentDirectory` класса `Environment`, позволяющее получить текущий каталог для данного приложения и доступное как для чтения, так и для записи. Среди других полезных свойств этого класса отметим `CommandLine`, возвращающее строку с полным именем запущенного exe-файла и следующими за ним параметрами командной строки, `SystemDirectory`, возвращающее полное имя системного каталога Windows, а также набор свойств, позволяющих получить информацию о компьютере, пользователе и операционной системе: `MachineName`,

UserName, OSVersion, Version (последнее свойство возвращает полный номер используемой версии .NET Framework). Есть также свойство-массив GetCommandLineArgs типа string[], первый элемент которого (с индексом 0) содержит полное имя exe-файла, а каждый последующий элемент — очередной параметр командной строки (таким образом, свойство-массив GetCommandLineArgs представляет собой строку CommandLine, разобранную на составные элементы).

Кроме того, класс Environment позволяет получить и изменить значения различных *переменных окружения*, определенных в операционной системе. Для этого он имеет методы GetEnvironmentVariables (возвращает массив всех переменных окружения в виде пары строк *имя–значение*), GetEnvironmentVariable(name) (возвращает строковое значение для переменной окружения с именем name или null, если указанная переменная окружения не существует) и SetEnvironmentVariable(name, val) (задает переменной окружения с именем name новое строковое значение val). Если указанная в методе SetEnvironmentVariable переменная name ранее не существовала, то она создается; если параметр val является пустой строкой или равен null, то существующая переменная name уничтожается.

2.3. Использование параметров командной строки

Измените метод Main в файле Program.cs (листинг 2.7).

Листинг 2.7. Новый вариант метода Main

```
static void Main(string[] args)
{
    Console.WriteLine("Программа DISKINFO\n");
    Console.WriteLine(" Disk Type          Size (K)      Free (K)");
    Console.WriteLine(new String('-', 40));
    if (args.Length == 0)
        DInfo(Environment.CurrentDirectory[0]);
    else
        foreach (string d in args)
            DInfo(d);
    Console.WriteLine("\nДля завершения программы нажмите <Enter>...");
    Console.ReadLine();
}
```

Результат. Если в качестве параметров командной строки указано одно или несколько имен дисков, то выводится информация об этих дисках (параметры командной строки должны отделяться друг от друга пробелами). Если параметры не указаны, то выводится информация о текущем диске. Для задания параметров командной строки в среде Visual C# следует выполнить команду **Project | DISKINFO Properties...**, в загруженном в редактор списке свойств проекта выбрать раздел **Debug** и ввести параметры командной строки в поле **Command line arguments**. Так, если в строке параметров указать **a d f z**, то в результате выполнения программы будет выведен текст, подобный приведенному в листинге 2.8.

В системе Windows программу с параметрами можно запустить, например, из меню **Пуск** командой **Выполнить...**:

```
C:\CSProjects\DISKINFO\bin\Debug\DISKINFO.exe a d f z
```

Если программа запускается с помощью *ярлыка* (shortcut), то ее параметры командной строки можно задать в окне свойств, которое вызывается командой **Свойства** контекстного меню ярлыка. В окне свойств надо перейти на вкладку **Ярлык** и указать нужные параметры в поле **Файл** (или **Объект**).


Листинг 2.8. Содержимое консольного окна при выполнении программы

Программа DISKINFO

```
Disk Type          Size (K)          Free (K)
=====
A   Removable      ---              ---
D   Fixed          16 370 256      4 285 224
F   Fixed          6 309 192       543 420
Z   ---            ---              ---
```

Для завершения программы нажмите <Enter>...

Ошибка. Если в качестве одного из параметров командной строки указать строку, начинающуюся не с латинской буквы, то при попытке обработать этот параметр возникнет *ошибка времени выполнения*, которая в случае запуска программы непосредственно в системе Windows приведет к аварийному завершению программы. Если программа была запущена из среды Visual C#, то произойдет возврат в среду Visual C# и в тексте программы будет выделен оператор, выполнение которого привело к возникновению ошибки. В последнем случае для завершения программы достаточно нажать комбинацию клавиш

<Shift>+<F5> или кнопку с изображением синего квадрата  (более подробно действия при возникновении ошибок времени выполнения описываются в разд. 3.1).

Исправление. В методе `DInfo` перед оператором

```
DriveInfo di = new DriveInfo(d);
```

добавьте следующий фрагмент:

```
if (d[0] < 'A' || d[0] > 'z')
    return;
```

Результат. Теперь программа обрабатывает только те параметры, которые начинаются с латинской буквы (прописной или строчной).

ПРИМЕЧАНИЕ

Другим способом исправления отмеченной ошибки является явная обработка возникающей исключительной ситуации (*исключения*) в блоке `try-catch` (см. главу 3). Однако если имеется возможность исправить ошибку без привлечения механизма обработки исключений, то этой возможностью следует воспользоваться, поскольку обработка исключений существенно замедляет выполнение программы.

Комментарий

Для определения количества параметров командной строки достаточно воспользоваться свойством `Length` массива `args` (см. листинг 2.7). Для перебора всех параметров мы воспользовались циклом `foreach`, на каждой итерации которого переменная `d` получает значение очередного элемента массива `args`. Заметим, что изменять элементы массива с помощью цикла `foreach` нельзя.